Meriedweg 11
CH-3172 Niederwangen
Switzerland
info@netmodule.com
http://www.netmodule.com
Tel +41 31 985 25 10
Fax +41 31 985 25 11

# User Manual JOPC-Bridge

Sort of document: User Manual
Version, date: **1.5**
Document order number: User Manual JOPC Bridge V4

Valid for JOPC-Bridge
Software Release: 4.6.0000

© 1998 - 2008 NetModule AG, Niederwangen, Switzerland
All Rights Reserved. Copying of this document or parts of it is prohibited.

# Table of contents


**1 About this user manual ..... 4**
1.1 The purpose of this manual ..... 4
**2 Service and Support ..... 5**
2.1 Technical Support ..... 5
2.2 Supply of Documentation and Utilities ..... 5
2.3 Feedback ..... 5
**3 JOPC-Bridge ..... 6**
3.1 Product Parts ..... 6
3.2 Java Class Overview JOPCBridge.jar ..... 7
3.2.1 Short Class Overview ..... 7
3.3 UML Class Diagram of JOPC-Bridge ..... 8
**4 Test client for Data Access ..... 9**
4.1 Overview ..... 9
4.2 Connect and show address range ..... 9
4.3 Add an item, read and write ..... 10
**5 Test client for Alarms & Events ..... 11**
5.1 Overview ..... 11
5.2 Connect, show categories and attributes ..... 11
5.3 Observance of the event sources ..... 12
**6 Sample program to access an OPC-DA server ..... 13**
6.1 Overview ..... 13
6.2 Information about the source code ..... 13
**7 Sample program to access an OPC AE server ..... 16**
7.1 Overview ..... 16
7.2 Information about the source code ..... 16
**8 OPC ..... 18**
8.1 General ..... 18
8.2 The OPC Data Access specification (OPC DA) ..... 18
8.2.1 The OPCServer class ..... 18
8.2.2 The OPCGroup class ..... 18
8.2.3 The OPCItem class ..... 19
8.3 The OPC Alarm und Events specification (OPC AE) ..... 20
**A Appendix: Document control ..... 22**


# Index of figures


Figure 1: Model of the JOPC-Bridge Architecture ..... 6
Figure 2: OPCDA Client, connect to OPC server ..... 10
Figure 3: OPCDA Client, add items, read and write ..... 10
Figure 4: OPCAE client, connect, show event categories and attributes ..... 11
Figure 5: OPCAE client, observance of Event Sources ..... 12
Figure 6: OPC Data Hierarchy ..... 18
Figure 7: OPC EventTypes ..... 20

# Copyright © 1998 - 2008 NetModule AG; All Rights Reserved

This document contains proprietary information of NetModule AG. No part of the work described herein may be reproduced. Reverse engineering of the hardware or software is prohibited and is protected by patent law.

This material or any portion of it may not be copied in any form or by any means, stored in a retrieval system, adopted or transmitted in any form or by any means (electronic, mechanical, photographic, graphic, optic or otherwise), or translated in any language or computer language without the prior written permission of NetModule AG.

The information in this document is subject to change without notice. NetModule AG makes no representation or warranties with respect to the contents herein and shall not be responsible for any loss or damage caused to the user by the direct or indirect use of this information. This document may contain information about third party products or processes. This third party information is out of influence of NetModule AG therefore NetModule AG shall not be responsible for the correctness or legitimacy of this information. If you find any problems in the documentation, please report them in writing by email to info@netmodule.com at NetModule AG.

While due care has been taken to deliver accurate documentation, NetModule AG does not warrant that this document is error-free.

NetModule MSOS, MSOSi, JPC, WebT, NetGate, NetBox and NetModule AG are trademarks and the NetModule logo is a service mark of NetModule AG.

All other products or company names mentioned herein are used for identification purposes only, and may be trademarks or registered trademarks of their respective owners.

The following description of software, hardware or process of NetModule AG or other third party provider may be included with your product and will be subject to the software, hardware or other license agreement.

# 1 About this user manual

## 1.1 The purpose of this manual

This document describes how to access data on OPC servers from Java via the JOPC-Bridge.

In order to perform a quick check whether you can access your OPC Server using the JOPC-Bridge, feel free to use NetModule's test clients. Chapter 4 describes how to use the Test client for Data Access whereas chapter 5 addresses the Test client for Alarms & Events ..

Chapter Sample program to access an OPC-DA server ("Sample program to access an OPC-DA server") and chapter Sample program to access an OPC AE server ("Sample program to access an OPC AE server") show how to access an OPC server with a Java application.

You can use these sample application as a basis for your own application.

# 2 Service and Support

## 2.1 Technical Support

The NetModule AG web site provides technical online support. This web site is continually updated with the newest solutions, application scenarios and FAQ

If you cannot solve your problem with the above mentioned information sources, please send an email to the following address:

support-jopc@netmodule.com

Please add to your email the following details:

- Your name, email address, telephone number, company name
- Name of the device, serial number, number of the software release
- Describe your problem
- Describe the environment in which the device is used (network, WAN) and the application scenario
- Describe your unsuccessful attempts to solve the problem

## 2.2 Supply of Documentation and Utilities

An electronic version of this user manual is available from the NetModule AG support page:

http://www.netmodule.com/en/support

## 2.3 Feedback

Comments to the JOPC-Bridge and the user manual are welcome to the following email address:

support-jopc@netmodule.com

# 3 JOPC-Bridge

## 3.1 Product Parts

For accessing an OPC-Server the following libraries are required:

- JOPCBridge.dll (library to access to C)
- JOPCBridge.jar (Java library for classes such as OpcItem, OpcServer, etc.)

**Model of the JOPC-Bridge Architecture**

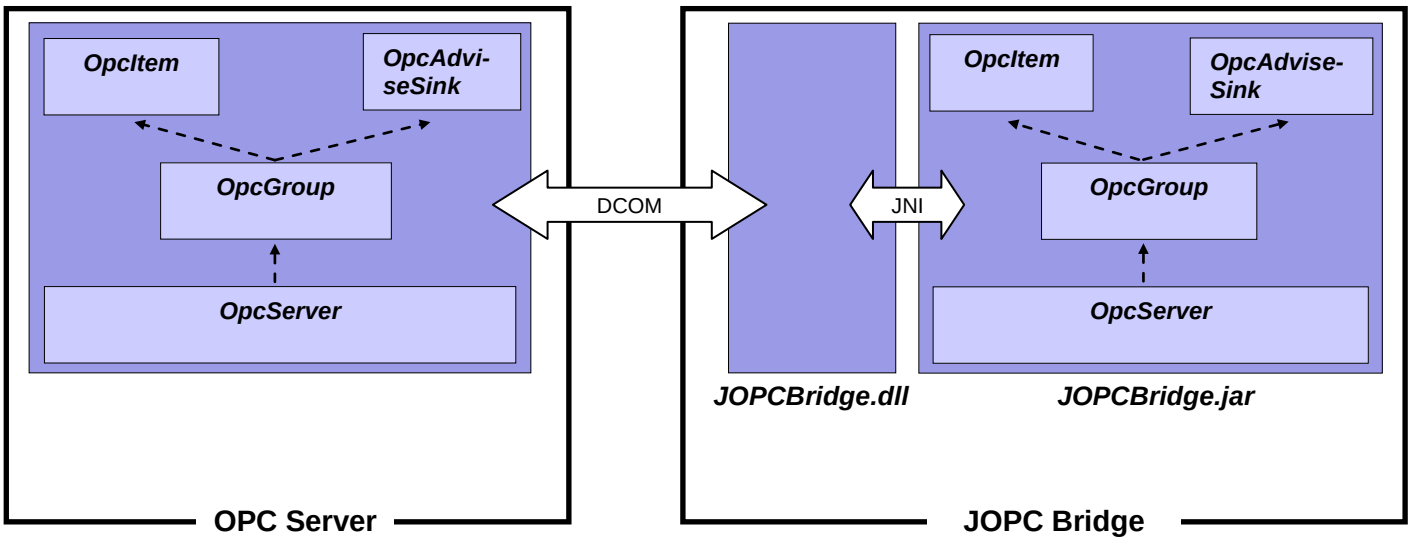



Figure 1: Model of the JOPC-Bridge Architecture

## 3.2 Java Class Overview JOPCBridge.jar

The classes are part of the package '**`com.netmodule.jpc.driver.opc`**'. For a detailed reference of the API, see the javadoc in the documentation folder. Extract the javadoc archive and open the file `index.html` with a Internet Browser of your choice.

### 3.2.1 Short Class Overview

`com.netmodule.jpc.driver.opc.OpcServer`

- Establish a connection to a OPC DA server
- Browse items
- Add and delete groups
- Obtain the server status

`com.netmodule.jpc.driver.opc.OpcGroup`

- Add and remove items
- Read and write items

`com.netmodule.jpc.driver.opc.OpcItem`

- <stores data of the items>

`com.netmodule.jpc.driver.opc.OpcAdviseSink`

- Event methods

`com.netmodule.jpc.driver.opc.OpcServerStatus`

- <stores data of the server status>

`com.netmodule.jpc.driver.opc.OpcEventServer`

- Establish a connection to an OPC AE server
- Setup an EventSubscription
- Query EventCategories and EventAttributes

`com.netmodule.jpc.driver.opc.OpcEventSink`

- Event methods

com.netmodule.jpc.driver.opc.OpcEventSubscription

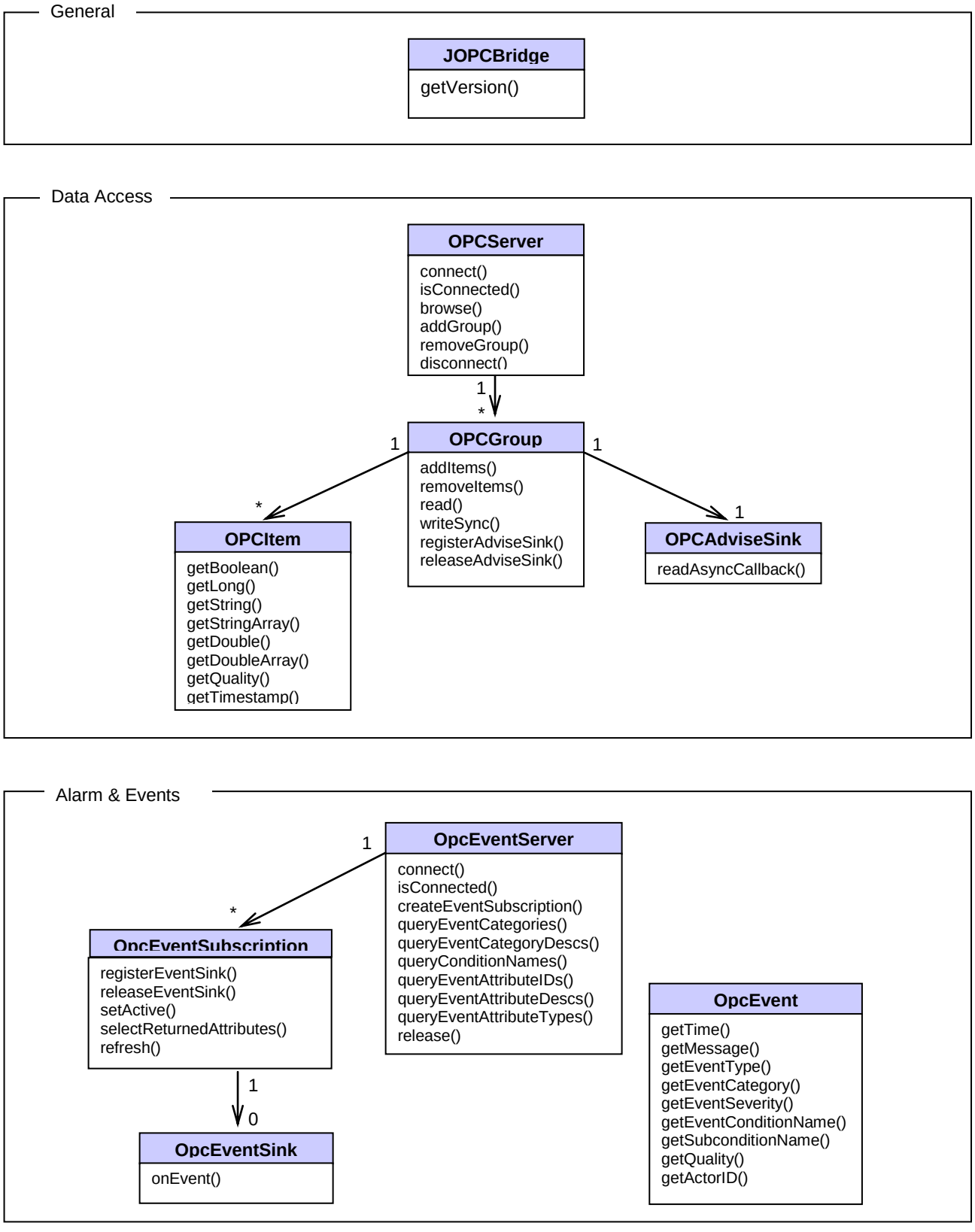
- Register and dissolve Events

com.netmodule.jpc.driver.opc.OpcEvent

- Contains the notification information for events occured
- Type, EventCategory, Severity …are accessible through properties

## 3.3 UML Class Diagram of JOPC-Bridge

For a detailed reference of the API, see the javadoc reference in the documentation folder.



# 4 Test client for Data Access

## 4.1 Overview

The tools folder contains a demo GUI that allows to access an OPC DA server through JOPC Bridge. This application consists of the files:

- OPCDAClient.bat – a batch file to start the OPC Data Access Demo Client
- Jopctools.jar – the compiled OPCDAClient class as java library
- ../lib/jopcbridge.jar – the java part of JOPC Bridge
- ../lib/jopcbridge.dll – the native part of JOPC Bridge

In order to start the OPC DA demo client, run the batch file `OPCDAClient.bat`. If the batch script cannot find your java environment, please perform one of the following steps:

- Add the java bin directory to your PATH environment variable or
- Set the JAVA_HOME environment variable to your JDK location, e.g. C:\j2sdk1.4.2_11

## 4.2 Connect and show address range

To connect to an OPC DA server the OPCDA Client requires the host and the server name of the OPC server. For remote hosts the host name (**\\hostxy** or IP-Address e.g. **\\172.66.22.22**) is entered in the corresponding field. For local OPC servers the host name is **\\localhost** or the field can be left empty.

If you do not know the server name on the host specified, simply enter the host, press Return and let the program try to obtain a list of all OPC servers installed.

Pushing “Browse Item” shows the address range. Then, the desired item can be selected from the list on the left. If the OPC server does not support “Browse Item” the name of the desired item can be entered directly in the field „ItemID“.

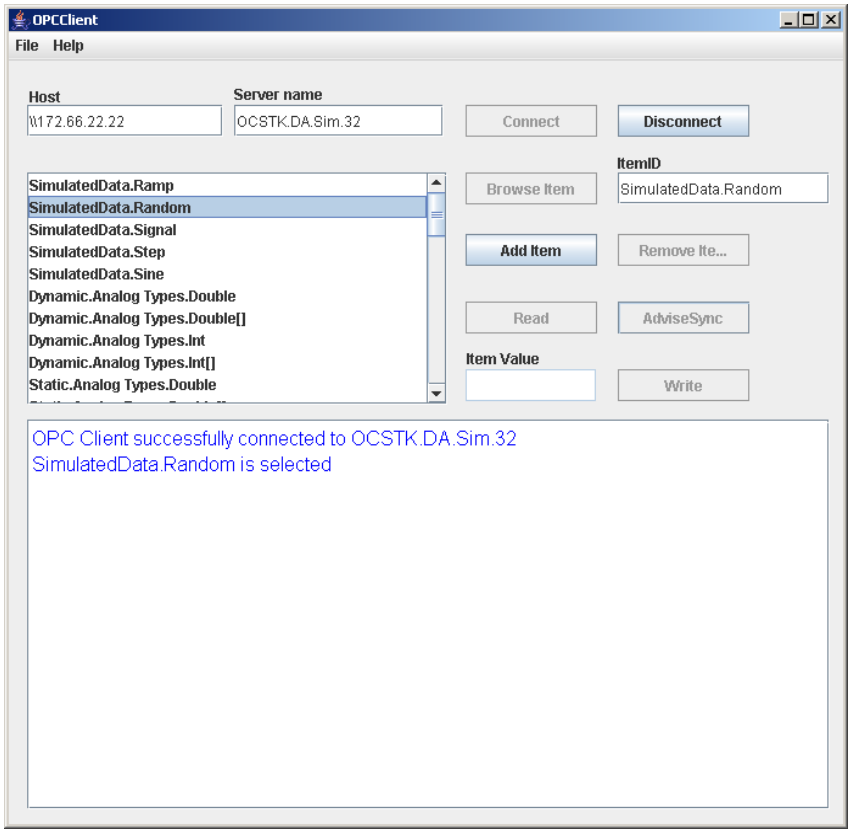


Figure 2: OPCDA Client, connect to OPC server

## 4.3 Add an item, read and write

Before and item can be read it must have been added to a group. The value of an item can be read one time by pushing “Read”. By setting “AdviseSink” the OPC client will be notified each time the value of the observed item changes. By pressing “AdviseSink” another time this notification will be disabled. The “Write” button is used to write a value of an item.

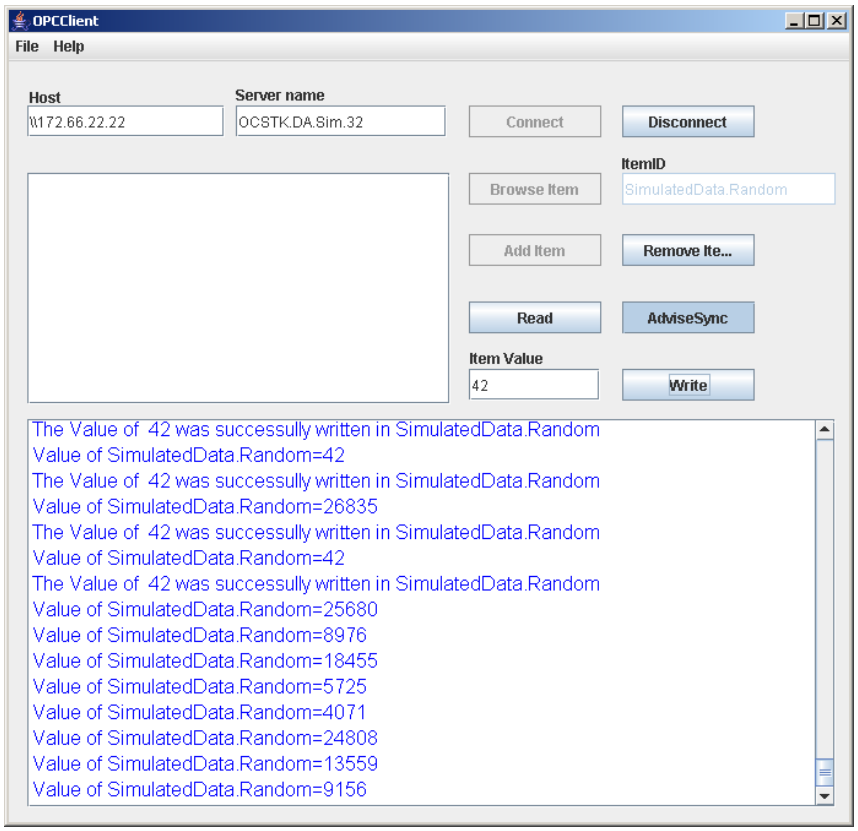



Figure 3: OPCDA Client, add items, read and write

# 5 Test client for Alarms & Events

## 5.1 Overview

The tools folder also contains a demo GUI that accesses an OPC AE server through JOPC Bridge. This application consists of the files:

- OPCAEClient.bat – a batch file to start the OPC Alarms & Events Demo Client
- Jopctools.jar – the compiled OPCAEClient class as java library
- ../lib/jopcbridge.jar – the java part of JOPC Bridge
- ../lib/jopcbridge.dll – the native part of JOPC Bridge

In order to start the OPC AE demo client, run the batch file `OPCAEClient.bat`. If the batch script cannot find your java, please perform one of the following steps:

- Add the java bin directory to your PATH environment variable or
- Set the JAVA_HOME environment variable to your JDK location, e.g. C:\j2sdk1.4.2_11

## 5.2 Connect, show categories and attributes

To connect to an OPCAE server the OPCAE client requires host and server name of the OPC server. For remote hosts the host name (**\\hostxy** or IP-Address e.g. **\\172.66.22.22**) is entered in the corresponding field. For local OPCAE servers the host name is **\\localhost** or the field can be left empty.

If you do not know the server name on the host specified, simply enter the host, press Return and let the program try to obtain a list of all OPC servers installed.

The various buttons can be used to show specific EventCategories and the corresponding EventAttributes.

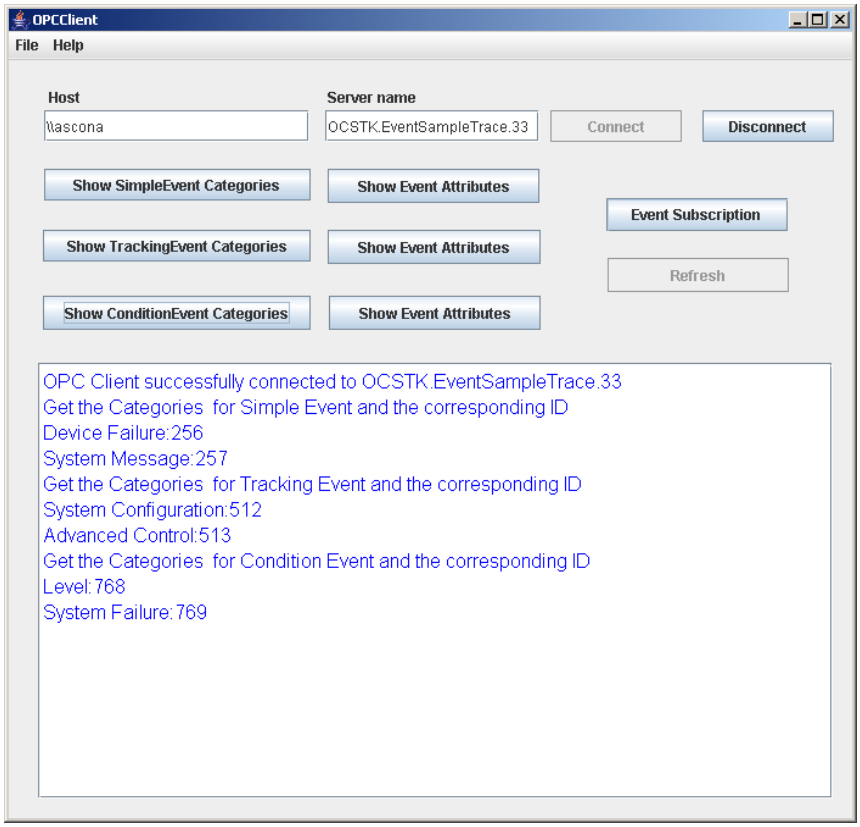



Figure 4: OPCAE client, connect, show event categories and attributes

## 5.3 Observance of the event sources

Event sources in the sever can be observed with “Event Subscription” If there is no automatic notification, the current values can be obtained by pressing “Refresh”

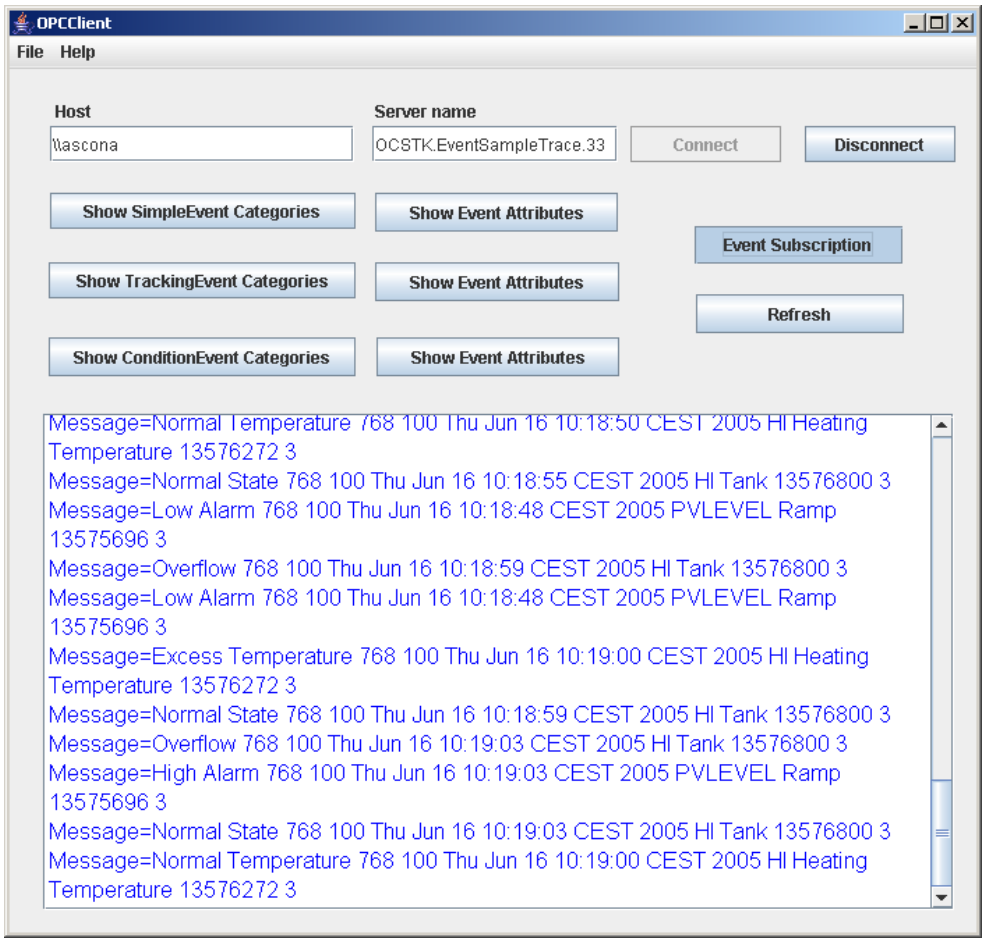

OPCClient
File Help
Host
\\ascona
Server name
OCSTK.EventSampleTrace.33
Connect
Disconnect
Show SimpleEvent Categories
Show Event Attributes
Event Subscription
Show TrackingEvent Categories
Show Event Attributes
Refresh
Show ConditionEvent Categories
Show Event Attributes
Message=Normal Temperature 768 100 Thu Jun 16 10:18:50 CEST 2005 HI Heating Temperature 13576272 3
Message=Normal State 768 100 Thu Jun 16 10:18:55 CEST 2005 HI Tank 13576800 3
Message=Low Alarm 768 100 Thu Jun 16 10:18:48 CEST 2005 PVLEVEL Ramp 13575696 3
Message=Overflow 768 100 Thu Jun 16 10:18:59 CEST 2005 HI Tank 13576800 3
Message=Low Alarm 768 100 Thu Jun 16 10:18:48 CEST 2005 PVLEVEL Ramp 13575696 3
Message=Excess Temperature 768 100 Thu Jun 16 10:19:00 CEST 2005 HI Heating Temperature 13576272 3
Message=Normal State 768 100 Thu Jun 16 10:18:59 CEST 2005 HI Tank 13576800 3
Message=Overflow 768 100 Thu Jun 16 10:19:03 CEST 2005 HI Tank 13576800 3
Message=High Alarm 768 100 Thu Jun 16 10:19:03 CEST 2005 PVLEVEL Ramp 13575696 3
Message=Normal State 768 100 Thu Jun 16 10:19:03 CEST 2005 HI Tank 13576800 3
Message=Normal Temperature 768 100 Thu Jun 16 10:19:00 CEST 2005 HI Heating Temperature 13576272 3

Figure 5: OPCAE client, observance of Event Sources

# 6 Sample program to access an OPC-DA server

## 6.1 Overview

The samples folder contains a demo GUI that accesses an OPC DA server through JOPC Bridge. This application consists of the files:

- OPCDASample.bat – a batch file to start the OPC DA sample program
- OPCDASample.java – the OPC DA sample program's source code
- Jopcsamples.jar – the compiled OPCDASample and OPCAESample class as java library
- ../lib/jopcbridge.jar – the java part of JOPC Bridge
- ../lib/jopcbridge.dll – the native part of JOPC Bridge

In order to start the OPC DA sample application, run the batch file `OPCDASample.bat`. But first some adjustments are necessary. Open the batch file with any text editor (e.g. notepad) and modify the parameters host and server according to your OPC server setup.

```
%JAVACMD% -Djava.library.path=%JAVA_LIBRARY_PATH% OPCDASample -
host=localhost -server=Matrikon.OPC.Simulation.1
```

If the batch script cannot find your java, please perform one of the following steps:

- Add the java bin directory to your PATH environment variable or
- Set the JAVA_HOME environment variable to your JDK location, e.g. C:\j2sdk1.4.2_11

## 6.2 Information about the source code

If the OPC-Server is located on a remote host its hostname (e.g. **\\hostxy**) or die IP address (e.g. **\\172.66.22.22**) is required.

```
OpcServer server =new OpcServer();
if (!server.connect(hostName, serverName)){
   System.out.println("couldn't connect to server :"+serverName);
   return;
}
System.out.println("Succesfull connected to server :"+serverName);
```

Read all ItemID's:

```
static String[] itemIDs;
itemIDs=server.getAllItemIDs();
for (int i=0;i<itemIDs.length;i++){
   System.out.println(itemIDs[i]);
}
```

If the OPC server does not support this feature an empty array will be returned. With server.getQueryOrganization() can be determined, how the server items are organized.

Add a group :

```
OpcGroup group=server.addGroup("MyGroup", 1);
```

The items can be accessed via the group. To achieve this, the items must be added to the group. For efficient access the client (this sample program) uses ClientHandles, which are provided by the server in OpcItem

In the sample program the browsed ItemIDs with an unequivocal ClientHandle are added to OpcGroup.

ServerHandles are used for reading and writing access. They are assigned by the OPC server and are provided by the method addItems().

```
int[] serverHandles = group.addItems(_itemIDs,clientHandles);
```

The server returns an array with ServerHandles which are needed for the access. Examples:

```
OpcItem[] items=group.read(OpcGroup.OPC_DS_DEVICE, __serverHandles);
if (group.writeSync(__serverHandles, items)==0)
```

Normally several values can be read and written at the same time. However many OPC server have troubles with error handling, therefore we suggest to only read or write one value at one time.

OpcAdviceSink informs about changes of values within an OpcGroup. For the distinction of the Items, client Handles are used. The method addItems() assigns those handles to the respective Item.

```
System.out.println("Install AdviceSink");
OpcAdviseSink opcAdviseSink=new OpcAdviseSink(group){
   public void readAsyncCallback(OpcItem item){
     if (item!=null)
        System.out.println("Value of "
        +itemIDs[item.getClientHandle()]+"="+item.toString());
     else
        System.out.println("unknown callback");
   }
   };
```

Getting values from OpcItem :

Getting a valid value from OpcItem be sure to use the corresponding method, because of type will not be checked. See following table:

| Variant Data Type | Method |
|---|---|
| VT_I1 | getByte() |
| VT_UI1 | getShort() |
| VT_I2 | getShort() |
| VT_UI2 | getInt() |
| VT_I4 | getInt() |
| VT_UI4 | getLong() |
| VT_I8 | getLong() |
| VT_R4 | getFloat() |
| VT_R8 | getDouble() |
| VT_BOOL | getBoolean() |
| VT_BSTR | getString() |
| VT_I1_ARRAY | getByteArray() |
| VT_UI1_ARRAY | getShortArray() |
| VT_I2_ARRAY | getShortArray() |
| VT_UI2_ARRAY | getIntArray() |
| VT_I4_ARRAY | getIntArray() |
| VT_UI4_ARRAY | getLongArray() |
| VT_I1_ARRAY | getLongArray() |
| VT_R4_ARRAY | getFloatArray() |
| VT_R8_ARRAY | getDoubleArray() |
| VT_BOOL_ARRAY | getBooleanArray() |
| VT_BSTR_ARRAY | getStringArray() |

# 7 Sample program to access an OPC AE server

## 7.1 Overview

The samples folder contains a demo GUI that accesses an OPC AE server through JOPC Bridge. This application consists of the files:

- OPCAESample.bat – a batch file to start the OPC AE sample program
- OPCAESample.java – the OPC AE sample program's source code
- Jopcsamples.jar – the compiled OPCDASample and OPCAESample class as java library
- ../lib/jopcbridge.jar – the java part of JOPC Bridge
- ../lib/jopcbridge.dll – the native part of JOPC Bridge

In order to start the OPC DA sample application, run the batch file `OPCAESample.bat`. But first some adjustments are necessary. Open the batch file with any text editor (e.g. notepad) and modify the parameters host and server according to your OPC server setup.

```
%JAVACMD% -Djava.library.path=%JAVA_LIBRARY_PATH% OPCAESample -
host=localhost -server=Matrikon.OPC.Simulation.1
```

If the batch script cannot find your java, please perform one of the following steps:

- Add the java bin directory to your PATH environment variable or
- Set the JAVA_HOME environment variable to your JDK location, e.g. C:\j2sdk1.4.2_11

## 7.2 Information about the source code

The client can query the supported EventCategories by the server. The description of the EventCategories and the respective number are returned.

```
String[] catDesc=eventServer.queryEventCategoryDescs(eventType);
int[]  cat=eventServer.queryEventCategories(eventType);
```

The identification number, the description and the Data types of the attributes of a specific category are examined.

```
int[] attrIDs=eventServer.queryEventAttributeIDs(eventCategory);
String[] attrDesc=eventServer.queryEventAttributeDescs(eventCategory);
short[] attrTypes=eventServer.queryEventAttributeTypes(eventCategory);
```

The connection to OPC AE is established over the OPC EventServer object:

```
OpcEventServer eventServer=new OpcEventServer();
OpcEventSubscription eventSubscription;
try {
  if (eventServer.connect(null, "FactorySoft.InProc.Alarm.1")!=0){
    System.out.println("Error connecting");
    return;
  }
}
System.out.println("new OpcEventServer()");
```

Over an EventServer an EventSubscription is established.

```
eventSubscription=eventServer.createEventSubscription();
```

Anonym class which overwrites the onEvent() methods of the class OpcEventSink. The OPCEventSink class is relevant for the transfer of the Event-Notification.

```
mySink=new OpcEventSink(){

  public void onEvent(OpcEvent event){
    System.out.println("Message="+ event.getMessage()+" "+
    event.getEventCategory()+" "+ event.getEventSeverity()+" "+
    event.getActiveTime()+" "+event.getConditionName()+"
    "+event.getCookie()+" "+event.getNumEventAttrs());

    if (event.getNumEventAttrs()>0){
      OpcVariantArray variantArray = event.getEventAttributes();
      for (int j=0;j<variantArray.getSize();j++){
        if (variantArray.isInt(j))
          System.out.println(variantArray.getInt(j));
          if (variantArray.isString(j))
            System.out.println(variantArray.getString(j));
          }
        }
      }
    }
  }
```

As soon as a relevant alarm or event occurs, the client is notified by the server and the following method is called (callback).

```
public void onEvent(OpcEvent[] events){
  for (int i=0;i<events.length;i++)
    onEvent(events[i]);
  }
};
```

Registration to start the EventNotification.

```
eventSubscription.registerEventSink(mySink);
```

# 8 OPC

## 8.1 General

General information about OPC is provided under http://www.opcfoundation.org.

## 8.2 The OPC Data Access specification (OPC DA)

Die OPC specification for Data Access divides the interfaces and its methods into three hierarchical sections

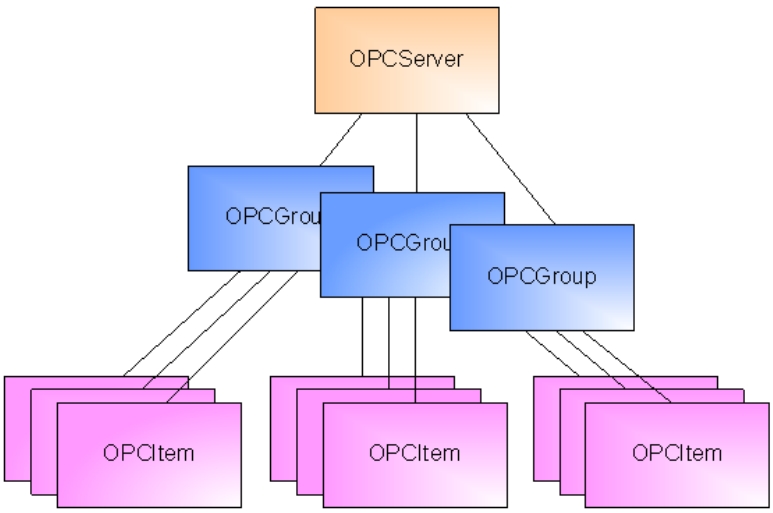



Figure 6: OPC Data Hierarchy

### 8.2.1 The OPCServer class

On the top there is the class OPCServer. An object of this class represents a manufacturer specific OPC-Server. The class has different methods and attributes that provide information about the status, the version and the address range of the available process variables of an OPC server object. Furthermore it manages the instances of the subordinate group OPCGroup.

### 8.2.2 The OPCGroup class

The class OPCGroup structures the process variables used by the OPC server. Means the OPCGroup objects an OPC client can form reasonable unities if process variables and execute action with them. For example all process variables of a screen of a visualization system can be combined in a group.

The class OPCGroup defines methods to read and write the process variables. Multiple variable can be combined within a task and handed over at the same time, specially when using an OPC server over a network (DCOM “Distributed COM) those summed calls make possible high speed execution.

### 8.2.3 The OPCItem class

An object of the class OPCItem represents a connection to a process variable. A process variable is an element within the address range of the OPC server, e.g. an input channel of a PLC.

An OPCItem is identified through its ItemID. This ItemID is name determined by the manufacturer of tbe server. It must be unique within a server address range.

Every OPCItem carries the attributes *value*, *quality* and *time stamp*. The quality of an OPCItem says whether the value of the variable could be determined surely (e.g. if there was a connection) and herby it determines the significance of the value. The time stamp says when the value of the process variable has last changed.

An Item can contain unrestricted data (e.g. value of a pressure or temperature or flow sensor, It can also contain different data types (e.g. Long, Boolean, String, etc.).

## 8.3 The OPC Alarm und Events specification (OPC AE)

OPC AE enables a standardized exchange of alarm and event information. For example a PLC generates an event that must be shown on a client (e.g. our OPCEventServerTest Program)

Alarms can be generated by sensors (e.g. end switch, limit-switch, and emergency cutout)

Events mostly are results of a control function (e.g. end of a production step, progress in a batch production etc.).

The specification model differentiates between things that happen (**simple events** and **tracking-related events**) and things that exist (**condition-related events, i.e. alarms**).

The occurrence of an alarm can be acknowledged. An object hierarchy has been defined, that a client can adapt the server to its requirements.

Within the server various interfaces and methods have to be implemented. After the start of the component the client gets access to an interface of the OPCEventServer object. After that the client can apply OPC-EventSubscription objects. The objects help to observe event sources and send respective message to the client at the occurrence of an event.

Each event has a specific type. The specification defines three types of events:

**condition-related event:** An example for a condition-related event would be a positive or negative deviation from temperature limit values.

**tracking-related event:** An example for a tracking-related would be the changing of a se point

**simple event:** An example for a simple event would be a device failure

For all of these three types there are **categories** (level, deviation or discrete for condition-related events or device failure for simple event).

Here the specification just defines a few possibilities. More categories can be defined bases on real application scenarios. For condition-related events the different categories have conditions and sub conditions. The specification already contains some proposals for categories, conditions and sub conditions. Events also have a **severity**. This value indicates the urgency of a condition or a sub-condition. The values are between 1 and 1000, whereas 1 possesses the lowest priority. A large variety of filters can be defined, e.g.

<Type=condition-related Events,
Category=Level, Severity=500>

In case of event occurrence, the server sends an event notification to the client which has assigned the event to an OPCEventSubscription object via a filter. Depending on the type of event the notification can have different numbers of parameters.

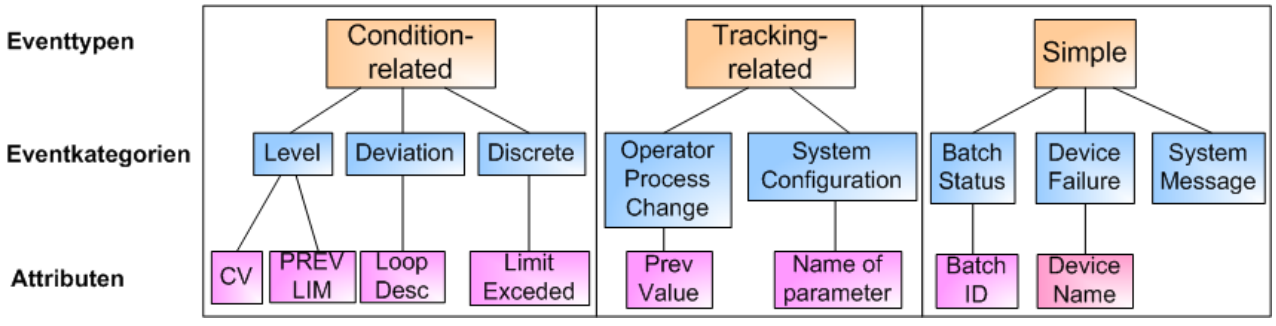



Figure 7: OPC EventTypes

During the notification about a simple event, values for the parameters Source (Information from the event space) Time (when did the event occur?), Type, Event Category, Severity and Message are transferred.

For notification about tracking-related event, there is the additional parameter ActorID. It contains a numerical identifier to indicate the cause of the event. The creation of the identifier and the meaning of the value are not described in the specification. The largest number of parameters must be transferred for notification about a condition-related event. The following parameters are added to the ones already mentioned:

**ConditionName:** Name of the condition from the event area.

**SubConditionName:** Name of the sub condition, if any.

**ChangeMask:** Indicates in what way the state of condition has changed, e.g. inactive -> active

**State: State** of the condition, e.g. active, acknowledged, …

**ConditionQuality:** The parameter can be compared with the quality of a value from the Data Access Specification. A notification is also sent if the state changes.

**AckRequired:** The event must be acknowledged.

**Active Time:** Indicates the time at which the state became active. This value is not identical with "Time" since the latter indicates the time of event occurrence. The receipt of an acknowledgement also leads to an event. After the receipt, the values of the parameters differ.

**Cookie:** Used by the client to acknowledge and by the server to relate the acknowledgement to the event.

There may be other attributes in addition to these mandatory ones. They have to be supported for all events of a category, i.e. they are defined for a category.[1]

[1] Praxis Profiline :OPC The Standard for Automation ISBN 3-8259-1917-7

# A Appendix: Document control

**Changes:**

| Version | Date | Name | Reason |
|---|---|---|---|
| 1.0 | 17.6.2005 | | Valid |
| 1.1 | 21.2.2006 | mme | ch. 2.1, e-Mail addresses corrected |
| 1.2 | 25.4.2006 | abt | ch. 4.1, 5.1, 6.1, 7.1 |
| 1.3 | 15.5.2006 | abt | ch. 3.1 new model of JOPC Bridge<br>ch. 4.1 obtain a list of OPC servers on host<br>ch. 5.1 obtain a list of OPC servers on host |
| 1.4 | 10.1.2007 | mme | ch. 6.2 getting value from OpcItem |
| 1.5 | 12.3.2008 | abt | ch. 6.2 completed table for OpcItem access |